Simultaneous Security of Bits in the Discrete Log.

René Peralta (*)

Computer Science Division

University of California

Berkeley, California.

ABSTRACT

We show that c log log P simultaneously secure bits can be extracted from the discrete log function. These bits satisfy the next-bit unpredictability condition of Blum and Micali. Therefore we can construct a cryptographically secure pseudo random number generator which produces c log log P bits per modular exponentiation under the assumption that the discrete log is hard.

1. Introduction.

Let $P = 2^S q + 1$ (q odd) be an odd prime and α a generator for the multiplicative group of integers modulo P. The problem of solving $\alpha^X = \beta \ (mod\ P\ )$ for X is called the **discrete log problem.** The fastest known algorithm for solving the discrete log runs in time $e^{\sqrt{\ln P \ln \ln P} + o(1)}$. (Coppersmith,) However, certain bits of X (for example the least significant bit) can be retrieved in polynomial time in log P . It is of theoretical and practical interest to identify the hard bits of X, as well as groups of bits which are hard simultaneously.

(*) Research sponsored in part by NSF grant MCS-82-04506

F. Pichler (Ed.): Advances in Cryptology - EUROCRYPT '85, LNCS 219, pp. 62-72, 1986.

We start by defining the concept of a secure single bit with respect to an underlying function f.

Definition 1 . A boolean predicate B(X) of X is **hard** with respect to a function f if an oracle which outputs B(X) on input f(X) can be used to invert f in polynomial time.

We now extend this notion to consider the simultaneous security of several bits. Call a boolean predicate trivial if it is identically 0 or identically 1.

Definition 2 . A k-bit predicate $B_k(X)$ is **hard** with respect to a function f if for every non-trivial boolean predicate B on k bits, an oracle which outputs $B\ (B_k(X))$ on input f(X) can be used to invert f in polynomial time. If B_k is a hard predicate then we say that bits $B_k(X)$ of X are **weak simultaneously secure.**

Blum and Micali (Blum, 1982) showed a hard boolean predicate for the discrete log. Long and Widgerson (Long, 1983) show that c log log P high order bits of X are weak simultaneously secure. Long (Long, 1984) shows that c log log P low order bits are also weak simultaneously secure.

Weak simultaneous security, however, is not the strongest possible notion of security. In particular, weak simultaneous security of k bits is **not enough** to use all k bits in a cryptographically secure pseudo random number generator.

The notion of next-bit unpredictability came up in the study of pseudo random number generators. Blum and Micali (Blum, 1982) showed the first pseudo random number generator which had this property. Yao (Yao, 1982) later showed that pseudo random number generators with this property pass all polynomial statistical tests for randomness. Below we define this notion outside the context of pseudo random number generators. In section 5 we show that next-bit unpredictability is stronger than weak simultaneous security in the sense that if k bits of X are next-bit unpredictable then they are also weak simultaneously secure.

Definition 3 . Let f be a function from Z_N to Z_N. k bits $x_1, \cdots, x_t$ of X are **next-bit**

unpredictable if for every l $(1 \leq l < k)$ an oracle which on input $f(X), x_1, \ldots, x_l$ outputs x_{l+1} on $\frac{1}{2} + \epsilon$ fraction of all inputs X, can be used to invert f in probabilistic polynomial time. (Here, $\epsilon > (log\ N)^{-c}$ for some constant c)

The main result in this paper is that if $P = 2^S q + 1$, with q an odd integer, then the $k = c \log \log P$ bits immediately following the s-th. least significant bit of X are next-bit unpredictable in the discrete log. Thus we can extract $c \log \log P$ bits per modular exponentiation in a pseudo random number generator based on the discrete log:

Let x_0 be a random number in Z_P. Let α be a generator for Z_P. Let $x_i = \alpha^{x_{i-1}} (mod\ P)$. Extracting the $c \log \log P$ bits immediately following the s-th. lsb. of $x_L, x_{L-1}, \ldots, x_0$, we obtain the **discrete log pseudo random sequence.**

Vazirani and Vazirani (Vazirani, 1984) have recently shown that $c \log \log P$ secure bits can also be extracted from the $x^2\ mod\ N$ generator of Blum, Blum, and Shub, (Blum, 1982) as well as from other encryption schemes based on factoring.

2. The S least significant bits of X are easy

In this section we show that the discrete log problem reduces to the problem of computing $\alpha^{2^{S-1}T} (mod\ P)$ from $\alpha^{2^S T} (mod\ P)$.

Pohlig Hellman (Pohlig, 1978) first gave an algorithm to compute the discrete log in the special case that $P = 2^S + 1$. In fact, their techniques show that the S least significant bits of X can be efficiently computed from $\alpha^X (mod\ P)$ where $P = 2^S q + 1$.

We use a slightly different method, introducing the technique of shifting X to the right by computing the square root of α^X. This technique will be used throughout this paper.

Square roots modulo a prime number are computable in probabilistic polynomial time. (Rabin, 1980) A quadratic residue modulo P is of the form $\alpha^{2t} (mod\ P)$. Therefore, if $\alpha^X = \beta (mod\ P)$, the least significant bit of X is 0 if and only if β is a quadratic residue. In this case the roots of β are $\alpha^{\frac{X}{2}} (mod\ P)$ and $\alpha^{\frac{X}{2} + \frac{P-1}{2}} = \alpha^{\frac{X}{2} + 2^{S-1}q} (mod\ P)$. The first of these

is called the **principal square root** of β (with respect to the generator α).

Blum and Micali (Blum, 1982) have shown that if we could compute the principal square root of β then we would be able to solve the discrete log in polynomial time: If β is a nonresidue we know that the lsb. of X is 1. We can set this bit to 0 by dividing β by α. Then we divide X by 2 by computing the principal square root. Thus we have shifted X to the right, moving X's 2nd. lsb. to the lsb. position, where it can be determined by testing quadratic residuosity. We can keep shifting until we obtain all bits of X. Thus we have shown the following:

(Blum-Micali) the discrete log reduces to the principal square root problem.

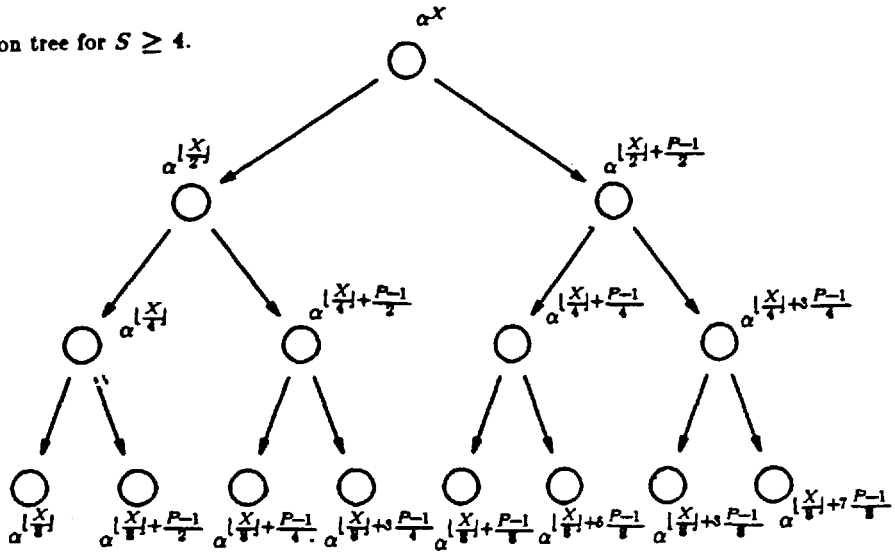
We cannot in general compute the principal square root of X. Notice, however, that if β is a quadratic residue, then both roots $\alpha^{\frac{X}{2}}$ and $\alpha^{\frac{X}{2}+2^{S-1}q}$ of β have the same quadratic character provided $S > 1$ i.e. the lsb. of the roots are equal. Choose an arbitrary root, set its lsb. to 0, and again compute a root of the result. This time there are four possible results , but provided $S > 2$ they all have the same quadratic character. We can in this manner compute the S least significant bits of X. The computation tree is shown below. Any path down this tree yields the correct bits. If we can compute these bits then we can set them to 0. Thus we have shown the following:

the discrete log reduces to solving the equation $\alpha^{2^S T} = \beta \, (mod \; P)$ **for T.**

Combining the two results we have:

the discrete log reduces to finding the principal square root $\alpha^{2^{S-1}T}$ **of** $\beta = \alpha^{2^S T} \; (mod \; P)$

Computation tree for $S \geq 4$.



Nodes at the same level have the same quadratic character.

3. The $s+1$ *st.* lsb. of X is a hard bit in the discrete log.

Suppose we have an oracle which on input P,α,β , outputs the $s+1$ *st.* lsb. of X. Then we can set this bit to 0 by dividing by α^{S+1} if necessary. Using the results of the previous section (and the oracle), the discrete log problem then reduces to finding the principal square root of $\beta = \alpha^{2^S T} \ (mod\ P)$ where T is even. But this is easy since the principal square root $\gamma = \alpha^{2^{S-1}T} \ (mod\ P)$ of β is the unique root which satisfies $\gamma^t = 1 \ (mod\ P)$. To see this recall that $\alpha^{P-1} = \alpha^{2^S t} = 1 \ (mod\ P)$. Then $\gamma^t = \alpha^{2^S t \frac{T}{2}} = 1 \ (mod\ P)$, whereas $(-\gamma)^t = -1 \ (mod\ P)$ since q is odd.

It will follow from Theorem 1 of the next section that this result holds even in the case where the oracle is correct in $\frac{1}{2} + \epsilon$ fraction of inputs. This result is included in (Long, 1984) along with a proof that almost all bits of X are hard with respect to oracles which are always correct.

4. $c \log \log P$ next-bit unpredictable bits

Let x_i be the ith. least significant bit of X.

Theorem 1. Let $k = c \log \log P$ for some constant c. Then $x_{s+1}, \cdots, x_{s+k}$ are next-bit unpredictable in the discrete log if we require the oracle to predict correctly on every input.

Proof: Suppose there exists l, $1 \leq l < k$, and an oracle O which on input $(P, \alpha, \beta, x_{s+1}, \cdots, x_{s+l})$ outputs x_{s+l+1}. As before, we may assume $X = 2^s T$. Algorithm I computes X in probabilistic polynomial time. Several operations are performed on the value of X. These can be done in polynomial time even when the value of X is not known but $\alpha^X \ (mod\ P\ )$ is. The operations are:

- test whether X equals a particular value.
- assignment (Y := X).
- division by 2 when the $S + 1$ least significant bits of X are known.
- setting a particular bit of X to 0 when the value of the bit is known.

Algorithm I - Solve $\alpha^X = \alpha^{2^S T} = \beta$ for X.

```
begin

Y := X ;
for every possible value of x_{s+1} ··· x_{s+l} do
begin
 i := 0;
 repeat until Y = 0 or i > log P

   begin
    i := i + 1;
    obtain bit s+l+1 of Y from the oracle;
    { assume this is also the s+l+i th. bit of X}
    set the s+1 st. bit of Y to 0
    Y := Y/2  { compute the principal square root of α^Y }
   end

 test the value constructed for X in the equation α^X = β (mod P ) ;
 if the equation holds then stop - X has been found
end

end.
```

Consider the iteration of the for-loop in which the correct value of $x_{s+1} \cdots x_{s+l}$ is

assumed. Recall that to compute the principal square root of α^Y, where $Y = 2^S T$, all we need to know is y_{s+1}. This bit is known correctly in the first iteration of the repeat loop. At each iteration of the repeat loop, the $s+1$ st. bit of Y is discarded and the higher order bits are shifted to the right by one position. The oracle allows the algorithm to see the bit which is shifted into th $s+l$ th. position. Thus , at each iteration , the algorithm knows bits $s+1$ through $s+l$ of Y, and in particular bit $s+1$ of Y .∇

Theorem 2 . Let $k = c \log\log P$ for some constant c. Then $x_{s+1}, \cdots, x_{s+k}$ are next-bit unpredictable in the discrete log.

Proof: Algorithm-I computes the discrete log using an oracle which is always correct. Now suppose the oracle is correct on $\frac{1}{2} + \epsilon$ fraction of inputs, with $\epsilon = \frac{1}{2^u}$ ($u = O(\log\log P)$). We will construct an oracle which is correct with probability exponentially close to 1 for all $X < \frac{P}{2^{u+1}}$. Note that, in the iteration of the for loop in which the correct value of $x_{s+1} \cdots x_{s+l}$ is assumed, the oracle is queried for monotonically decreasing values of Y. Therefore, for the algorithm to work, we need only **start** with an initial value of X which is less than $\frac{P}{2^{u+1}}$.

Note that Algorithm-I always knows the $s+l$ least significant bits of X. Therefore, if $X < \frac{P}{2^{u+1}}$ then x_{s+l+1} can be determined from the $s+l+1$ st. bit of $X + r$ provided $0 \leq r < P\frac{2^{u+1}-1}{2^{u+1}}$, since then $X + r < P$. This gives us a way of randomizing queries to the oracle. To determine the $s+l+1$ st. bit of X we query the oracle on the $s+l+1$ st. bit of X+r for random values of r in the specified range. We now show that the probability of obtaining a correct answer on each such random query is $\geq \frac{1}{2} + \frac{3\epsilon}{4}$.

Let $S = \{X + r \ / \ 0 \leq r < P\frac{2^{u+1}-1}{2^{u+1}}\}$. Notice $|S| = P\frac{2^{u+1}-1}{2^{u+1}}$, and every element of S is less than P provided $X < \frac{P}{2^{u+1}}$.

Let $\frac{1}{2} + \hat{\epsilon}$ be the fraction of elements in S for which the oracle is correct. Then the total number of correct answers of O is less than or equal to the number of correct answers in S plus the cardinality of the complement of S. Thus

$$P(\frac{1}{2} + \epsilon) \leq (\frac{1}{2} + \hat{\epsilon})|S| + P - |S| \quad ==>$$

$$P(\epsilon - \frac{1}{2}) \leq |S|(\hat{\epsilon} - \frac{1}{2}) \quad ==>$$

$$\hat{\epsilon} \geq \frac{P}{|S|}(\epsilon - \frac{1}{2}) + \frac{1}{2} = \frac{2 - 2^u}{2^{u+1} - 1} + \frac{1}{2} = \frac{3}{2}(\frac{1}{2^{u+1} - 1}) \geq \frac{3}{2}(\frac{1}{2^{u+1}}) = \frac{3\epsilon}{4}.$$

Thus the oracle is correct on $\frac{1}{2} + \frac{3\epsilon}{4}$ fraction of all elements in the set S. Therefore, by querying the oracle on a polynomial number (in log P) of points we obtain the $s + l + 1$ st. least significant bit of X with negligible probability of error.

A problem remains in that we have assumed that $X < \frac{P}{2^{u+1}}$. This is solved by randomizing X i.e. we try to solve the equation $\alpha^{(X+R) \bmod (P-1)} = \hat{\beta} = \beta\alpha^R \ (mod\ P\ )$ for random values of R ($0 \leq R < P - 1$). With probability $\frac{1}{2^{u+1}}$, $(X + R)\ mod\ (P-1) < \frac{P}{2^{u+1}}$. Alternatively, we could simply try all possible values of the $u + 1st.$ most significant bits of X, setting these bits to 0 by dividing by the appropiate power of α. Thus our algorithm computes X in probabilistic polynomial time .▽

5. Next bit unpredictability implies weak simultaneous security

The next theorem shows that next bit unpredictability is a stronger notion than weak simultaneous security. Although this result is implied by a fundamental theorem of Yao, (Yao, 1982) it is included here because it has a straightforward proof.

Theorem 3 . Let f be a function from Z_N to Z_N. If $k = c \log\log N$ bits are next-bit unpredictable with respect to f then they are also weak simultaneously secure with respect to f.

Proof: Suppose bits $(x_1, \cdots, x_k)$ are next-bit unpredictable. Let B be a non-trivial predicate on

$(x_1, \cdots, x_k)$. Let O be an oracle for B given f(X). Let T be the set of values of $(x_1, \cdots, x_k)$ for which $B(x_1, \cdots, x_k) = 1$. Since B is non trivial , there exists a prefix $\vec{u} = u_1 \cdots u_l$ (possibly the empty string λ) for which the number of elements in T with prefix $\vec{u}1$ is distinct from the number of elements in T with prefix $\vec{u}0$. Assume, without loss of generality, that T contains more elements with prefix $\vec{u}1$ than with prefix $\vec{u}0$.

Let $\vec{x} = (x_1, \cdots, x_l)$. We make the simplifying assumption that all values of $(x_1, \cdots, x_k)$ are equally probable when X is random. Then, if $\vec{x} = \vec{u}$, the probability that O outputs x_{l+1} on input f(X) is $> \frac{1}{2}$. We construct an oracle $\hat{O}$ for x_{l+1} given $(f(X), x_1, \cdots, x_l)$ as follows:

$\hat{O}$: *If* $\vec{x} = \vec{u}$ *then output* $O(f(X))$ *else output the flip of a fair coin.*

Now we show $\hat{O}$ is correct on at least $\frac{1}{2} + (log\ N)^{-c}$ fraction of all inputs. Thus, by the next-bit unpredictability assumption we can use $\hat{O}$ (hence O) to invert f in polynomial time.

Let ρ be the probability that $\hat{O}$ is correct when X is chosen at random. Then

$$\begin{aligned}
\rho &= Prob.(\vec{x} \neq \vec{u}) * \frac{1}{2} + Prob.(\vec{x} = \vec{u}) * Prob.(\hat{O}(f(X), \vec{x}) = x_{l+1} \mid \vec{x} = \vec{u}) \\
&= (1 - 2^{-l}) * \frac{1}{2} + 2^{-l} * Prob.(\hat{O}(f(X), \vec{x}) = x_{l+1} \mid \vec{x} = \vec{u}) \\
&= (1 - 2^{-l}) * \frac{1}{2} + 2^{-l} * Prob.(O(f(X)) = x_{l+1} \mid \vec{x} = \vec{u}) \qquad (*)
\end{aligned}$$

There are 2^{k-l} elements $x_1, \cdots, x_k$ with prefix $\vec{u}$, and we know the fraction of these elements for which O outputs x_{l+1} is $> \frac{1}{2}$. Therefore $Prob.(O(f(X)) = x_{l+1} \mid \vec{x} = \vec{u}) \geq \frac{2^{k-l-1} + 1}{2^{k-l}}$.

Substituting in (*) we get

$$\begin{aligned}
\rho &\geq (1 - 2^{-l}) * \frac{1}{2} + 2^{-l} * \frac{2^{k-l-1} + 1}{2^{k-l}} \\
&= \frac{1}{2} + \frac{1}{2^k} = \frac{1}{2} + (logN)^{-c}. \nabla
\end{aligned}$$

Acknowledgements:

Much of the inspiration for this work comes from a wonderful course in cryptography taught by Clauss Schnorr at Berkeley in the Fall of 84.

I also wish to thank Manuel Blum and Umesh Vazirani for their constructive criticism and support.

References

Blum,.L. Blum, M. Blum, and M. Shub, "A Simple Secure Pseudo-Random Number Generator," *CRYPTO 82*, 1982.

Blum,.M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo Random Bits," *23rd. FOCS*, pp. 112-117, 1982.

Coppersmith,.

Coppersmith, "Unpublished Result," *Private Communication through C.P. Schnorr.*

Long,.D. Long and A. Widgerson, "How Discreet is the Discrete Log," *15th. STOC*, 1983.

Long,.D. Long, "The Security of Bits in the Discrete Logarithm," *PhD Dissertation*, Princeton University, January, 1984.

Pohlig,.

S. Pohlig and M. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance.," *IEEE Transactions on Information Theory*, vol. 1, no. 1, January 1978.

Rabin,.

M. Rabin, "Probabilistic Algorithms in Finite Fields," *Siam J. Comp.* , vol. 9, pp. 273-280, 1980.

Vazirani,.

U. Vazirani and V. Vazirani, "Efficient and Secure Pseudo Random Number Generation," *Proceedings of the 25th. FOCS* , 1984.

Yao,. A. Yao, "Theory and Applications of Trapdoor Functions," *1982 FOCS*, 1982.